Beyond Polyhedral Analysis of OpenStream Programs

Nuno Miguel Nobre nunomiguel.nobre@manchester.ac.uk

Joint work with: Andi Drebes, Graham Riley and Antoniu Pop

IMPACT 2019: January 23, 2019 | Valencia, Spain
Task-parallel streaming dataflow models have strong assets:

• Point-to-point synchronization
  ▪ Hide latency

• Numerous opportunities for parallelism
  ▪ Task, data and pipeline

• Scheduling is the runtime’s job

• Provide functional determinism
How to exploit today’s machines efficiently?

Task-parallel streaming dataflow models have strong assets:

- Point-to-point synchronization
  - Hide latency
- Numerous opportunities for parallelism
  - Task, data and pipeline
- Scheduling is the runtime’s job
- Provide functional determinism

But also disadvantages:

- Manually specified tasks
  - Challenging dependency specification
  - Hard debugging
  - What’s the right granularity?
- Memory footprint: no in-place writes
How to exploit today’s machines efficiently?

Task-parallel streaming dataflow models have strong assets:

- Point-to-point synchronization
  - Hide latency
- Numerous opportunities for parallelism
  - Task, data and pipeline
- Scheduling is the runtime’s job
- Provide functional determinism

But also disadvantages:

- Manually specified tasks
  - Challenging dependency specification
  - Hard debugging
  - What’s the right granularity?
- Memory footprint: no in-place writes
Why the polyhedral model?

- Arbitrarily compose loop transformations inc. tiling $\rightarrow$ **granularity control**
- Static program analysis $\rightarrow$ **streams memory footprint/bounding**
- Multi-objective: parallelism, **vectorization**, multi-level cache reuse
- Compact program representation unlike graph algorithms
- Despite restrictions: **stencils**, dense linear algebra and image filters
1) Manual granularity tuning
   - Motivating example: Gauss-Seidel stencil

2) Stream bounding & automatic granularity tuning
   - The polynomial indexing problem
   - Future work solutions
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

Tasks access stream elements through sliding **windows**:

![Diagram showing tasks and streams with sliding windows](image-url)
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

Tasks access stream elements through sliding **windows**: 

![Diagram of tasks accessing stream elements through sliding windows.](image-url)
Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

Tasks access stream elements through sliding **windows**:

Stream accesses dictate the dependencies between tasks
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

Tasks access stream elements through sliding **windows**:

![Diagram showing tasks and streams]

Stream accesses dictate the dependencies between tasks
OpenStream: a (very) short overview

Data-flow extension to OpenMP

- **Tasks**: units of work spawned as concurrent coroutines
- **Streams**: unbounded channels for communication between tasks

Tasks access stream elements through sliding **windows**:

Stream accesses dictate the dependencies between tasks.
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

![Diagram of 1D Gauss-Seidel iteration](image)
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

Diagram:
- Previous iteration
- Current iteration
- Current grid point
- Not yet computed
- Flow dependence distance vector
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

---

**Diagram:**
- `i`: Index
- `j`: Index
- Previous iteration
- Current iteration
- Current grid point
- Not yet computed
- Flow dependence distance vector
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

![Diagram](image-url)
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

![Diagram showing the Gauss-Seidel algorithm](image)
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
  for (j = 1; j < N - 1; ++j)
    phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

![Diagram](image-url)
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

![Diagram showing iteration process and notation](attachment:image.png)
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

Diagram showing iteration progression and dependency vectors.
1D Gauss-Seidel: stencil code granularity tuning

Sequential C [SeqC]

```c
for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
```

OpenStream: Fine-grained tasks [OS-FG]

```c
stream_array S[N];

for (i = 0; i < I; ++i)
    for (j = 1; j < N - 1; ++j)
        task {
            read once from S[j];  // phi[j] (discarded)
            peek once from S[j - 1]; // phi[j - 1]
            peek once from S[j + 1]; // phi[j + 1]
            write once into S[j];  // phi[j]

            // work function:
            // phi[j] = (phi[j - 1] + phi[j + 1]) / 2;
        }
```
1D Gauss-Seidel: stencil code granularity tuning

1) Semantically equivalent C code (SA)
2) Pluto source-to-source compiler
3) OpenMP parallel code [OMP-PT]
4) OpenStream: Pluto-tiled tasks [OS-PT]
1D Gauss-Seidel: stencil code granularity tuning

1) Semantically equivalent C code (SA)
2) Pluto source-to-source compiler
3) OpenMP parallel code [OMP-PT]
4) OpenStream: Pluto-tiled tasks [OS-PT]
1D Gauss-Seidel: results

\[ N = 32769, I = 32768 \]

\[
\begin{array}{|c|c|c|c|}
\hline
\hline
2T & \begin{array}{c} 2^6 \times 2^6 \end{array} & \begin{array}{c} 2^{13} \end{array} & \begin{array}{c} 2^2 \times 2^2 \end{array} \\
4T & \begin{array}{c} 2^7 \times 2^7 \end{array} & \begin{array}{c} 2^{12} \end{array} & \begin{array}{c} 2^2 \times 2^2 \end{array} \\
8T & \begin{array}{c} 2^8 \times 2^8 \end{array} & \begin{array}{c} 2^{11} \end{array} & \begin{array}{c} 2^3 \times 2^3 \end{array} \\
\hline
\end{array}
\]

---

**Bar Graph**

- **[SeqC]**
  - 13.33

- **[OS-FG]**
  - 7.41
  - 3180.89
  - 3016.67

- **[OS-PT]**
  - 4.30
  - 2.61
  - 1.88

- **[OS-ST]**
  - 3.56
  - 1.85
  - 1.00

- **[OMP-PT]**
  - 2.74
  - 1.48
  - 1.31

---

8 / 15
2D Gauss-Seidel: a visual picture

OpenStream: Fine-grained tasks [OS-FG]

- Previous iteration
- Current iteration
- Current grid point
- Not yet computed
- Flow dependence
- Distance vector
2D Gauss-Seidel: a visual picture

OpenStream: Pluto-tiled tasks [OS-PT]

- Loop iteration/fine-grained task
- Previous iterations of the outer loop
- Loop tile/Pluto-tiled task
2D Gauss-Seidel: a visual picture

OpenStream: Pluto-tiled tasks [OS-PT]

OpenStream: Spatially tiled tasks [OS-ST]
2D Gauss-Seidel: results

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
 \text{N x N} & \text{SeqC} & \text{OS-PT} & \text{OS-ST} & \text{OMP-PT} \\
\hline
2T & $2^5 \times 2^5 \times 2^5$ & $2^8 \times 2^8$ & $2^2 \times 2^2 \times 2^2$ & \\
4T & $2^5 \times 2^5 \times 2^5$ & $2^8 \times 2^8$ & $2^2 \times 2^2 \times 2^2$ & \\
8T & $2^5 \times 2^5 \times 2^5$ & $2^7 \times 2^7$ & $2^3 \times 2^3 \times 2^3$ & \\
\hline
\end{tabular}
\end{table}
The polynomial problem

- Stream indexing is polynomial
  - e.g. parametric tiling
The polynomial problem

• Stream indexing is polynomial
  ▪ e.g. parametric tiling

• Deadlock undecidability
  ▪ Albert Cohen, Alain Darte, and Paul Feautrier. 2016. Static Analysis of OpenStream Programs
The polynomial problem

- Stream indexing is polynomial
  - e.g. parametric tiling
- Deadlock undecidability
  - Albert Cohen, Alain Darte, and Paul Feautrier. 2016. Static Analysis of OpenStream Programs
- Schedule found: no deadlock
Future work: bounding streams

Dataflow task graph
Future work: bounding streams

Back-pressure dependencies

- Dataflow task graph: new edges (cycle)
- Poly. model: “just” new schedule restrictions (no schedule)

3-element stream: deadlock
Future work: bounding streams

Back-pressure dependencies

Dataflow task graph: new edges (cycle)
Poly. model: “just” new schedule restrictions (no schedule)

If schedule found: OpenStream’s runtime can schedule the program
Future work: coarsening task graphs

Dataflow task graph

- $t_0$
- $t_1$
- $t_2$
- $t_3$
Future work: coarsening task graphs

Dataflow task graph

Arbitrary coarsening:
deadlock
Future work: coarsening task graphs

Dataflow task graph

Loop strip-mining, facilitated by stream mushing

Arbitrary coarsening: deadlock

E.g. coalescing instances of the same task
Future work: coarsening task graphs

Dataflow task graph

Arbitrary coarsening: deadlock

e.g. coalescing instances of the same task

Loop strip-mining, facilitated by stream mushing

If schedule found: OpenStream’s runtime can schedule the program
• Task-parallel dataflow programs can benefit from polyhedral transformations
• Analyses and transformations are hindered by polynomials
• Bounding streams: adding back-pressure dependencies and finding a schedule
• Granularity control: loop strip-mining? how do we align this w/ current techniques?